OBJECT LOCALIZATION AND DETECTION

Contributeurs :

- Ahmed BEJAOUI.
- Aymen MEJRI.
- Mohamed Rostom GHARBI.

Encadré par :

- Florence D'ALCHE-BUC.

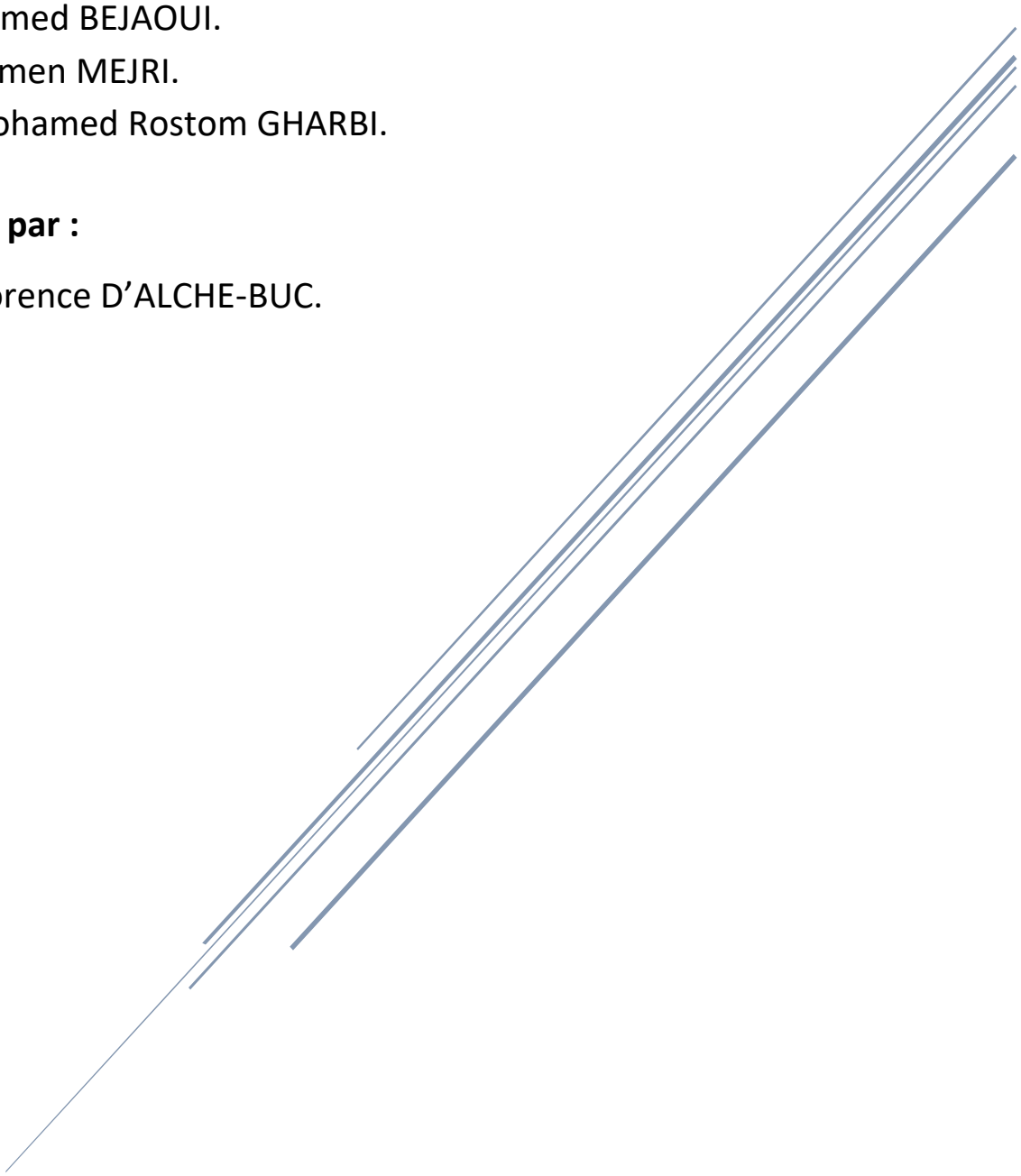

Table des matières

Table de figures

I. Introduction :

Pour cet article, nous allons essayer différentes méthodes pour adresser la problématique de localisation et de détection des objets dans les images.

Pour cela, nous allons utiliser plusieurs modèles en expliquant chacun d'entre eux, et après, faire la comparaison entre ces modèles.

Computer Vision est un domaine interdisciplinaire qui a eu un énorme essor ces dernières années (depuis que le CNN existe). La détection d'objet est une autre partie intégrante de la computer vision.

La différence entre les algorithmes de détection d'objets et les algorithmes de classification est que pour les algorithmes de détection, nous essayons de dessiner un cadre de sélection autour de l'objet d'intérêt pour le localiser dans l'image. En outre, vous ne pouvez pas dessiner qu'un seul cadre de sélection dans un cas de détection d'objet, il peut y avoir plusieurs cadres de sélection représentant différents objets d'intérêt dans l'image que vous ne savez pas le nombre d'avance.

La principale raison pour laquelle nous ne pouvons pas résoudre ce problème en construisant un CNN standard suivi par une couche entièrement connectée, est que la taille de la couche de la sortie est variable - non constante, car le nombre d'occurrences des objets d'intérêt est non fixé.

Une approche naïve pour résoudre ce problème consisterait à prendre différentes régions d'intérêt de l'image et à utiliser un CNN pour classifier la présence de l'objet dans cette région. Le problème de cette approche est que les objets d'intérêt peuvent avoir des emplacements spatiaux différents dans l'image et des formats d'image différents.

Par conséquent, il vous faudrait sélectionner un très grand nombre de régions et ceci pourrait être trop lent. Par conséquent, des algorithmes tels que **R-CNN**, etc. ont été développés pour trouver ces occurrences et trouver les objets rapidement.

II. Modèles :

2.1 RCNN :

2.1.1 Introduction & partie théorique du RCNN :

Pour contourner le problème de la sélection d'un grand nombre de régions, Ross Girshick et al. ont proposé une méthode utilisant la recherche sélective pour extraire seulement 2000 régions de l'image et ils les ont appelées propositions de régions (region proposals). Par conséquent, au lieu d'essayer de classer un très grand nombre de régions, nous pouvons simplement travailler avec 2000 régions.

Ces propositions de 2000 régions sont générées à l'aide de l'algorithme de recherche sélective qui est écrit ci-dessous.

Recherche sélective :

1. Générer une sous-segmentation initiale, nous générons de nombreuses régions candidates.
2. Utilisez un algorithme glouton pour combiner de manière récursive des régions similaires en de plus grandes.
3. Utiliser les régions générées pour produire les propositions de régions candidates finales.

Pour avoir plus d'information sur la recherche sélective, veuillez voir ce lien :

https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013/UijlingsIJCV2013.pdf.

Pour mieux comprendre le fonctionnement de la RCNN, nous présentons ci-dessous son architecture :

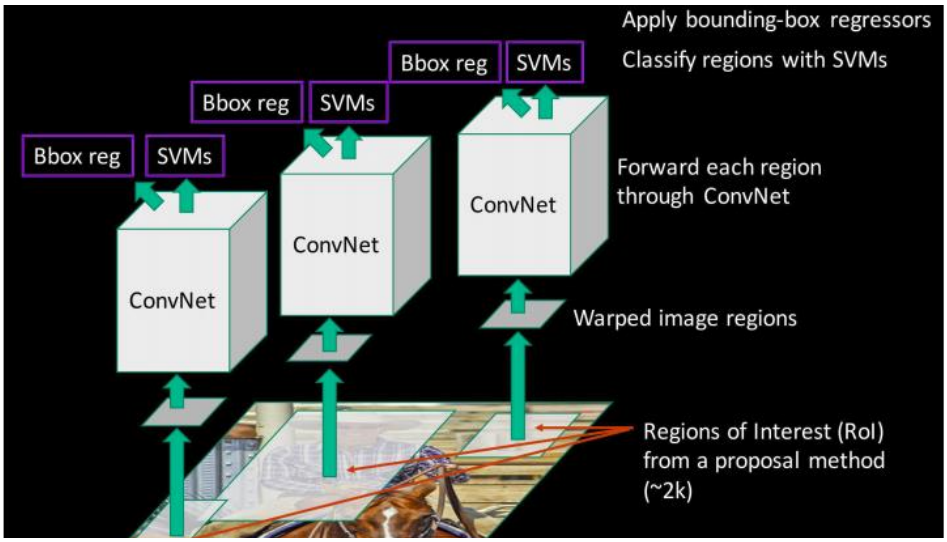

Figure 1: Architecture RCNN

2.1.2 Commentaires sur le RCNN :

Le problème de RCNN est qu'il n'est pas fait pour être rapide, il prend énormément de temps pour former le réseau, car il faudrait classer 2 000 propositions de région par image.

D'autre part, l'algorithme de recherche sélective est un algorithme fixe. Par conséquent, aucun apprentissage n'a lieu à ce stade. Cela pourrait conduire à la génération de mauvaises propositions de régions candidates.

RCNN, ne peut pas être utiliser pour des problèmes en temps réel vu qu'il prend approximativement 47 secondes pour chaque image du test set.

En effet, pour entrer plus dans les détails et pour comprendre pourquoi RCNN est aussi lent, nous présentons ci-dessous les étapes pour entrainer le modèle :

1. On commence par choisir un CNN pré-entrainé (AlexNet par exemple).

2. Entraînez à nouveau la dernière couche qui est entièrement connecté, avec les objets à détecter + la classe "pas d'objet".

3. Nous obtenons par suite les différentes propositions par image (~ 2k), que nous devons redimensionnez pour qu'elles correspondent à l'input du CNN.

4. Entrainer le SVM pour faire la classification binaire entre Object et background.

5. Régression pour les 4 coordonnées du bounding box.

2.2 Fast RCNN :

2.2.1 Partie théorique du Fast RCNN :

Le même auteur du R-CNN a résolu certains des inconvénients du R-CNN pour créer un algorithme de détection d'objet plus rapide, nommé Fast R-CNN.

L'approche est similaire à l'algorithme R-CNN, mais, au lieu de transmettre les propositions de région au CNN, nous transmettons l'image d'entrée au CNN afin de générer des features maps convolutives.

À partir de ces derniers, nous identifions la région des propositions. À l'aide d'une couche RoI, nous les transformons en une taille fixe afin de les insérer dans une couche entièrement connectée. À partir du vecteur d'entités RoI, nous utilisons une couche softmax pour prédire la classe de la région proposée ainsi que les valeurs de décalage pour le cadre de sélection.

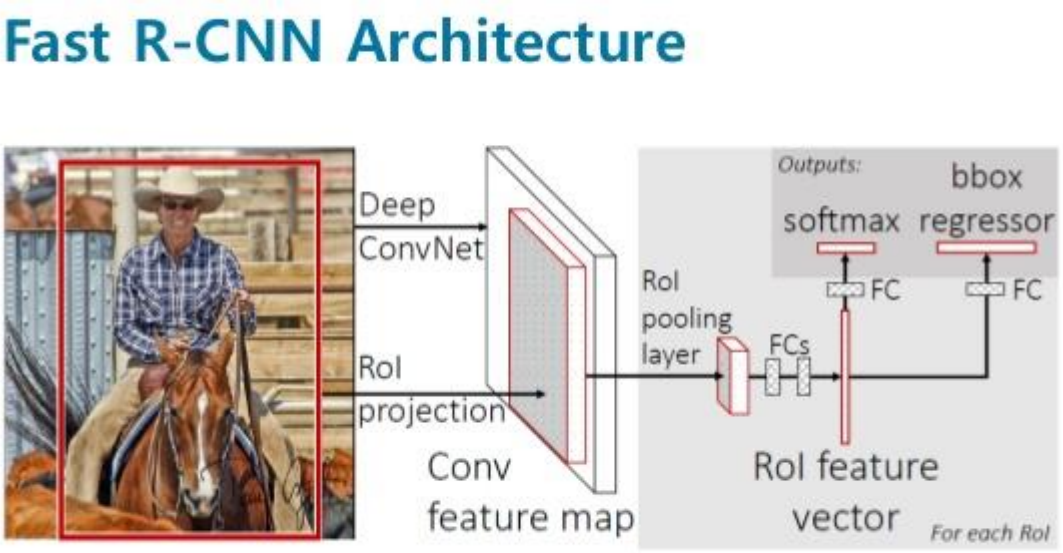

Figure 2 : Architecture Fast RCNN

Pourquoi la couhe RoI existe et qu'est-ce que c'est ?

Il s'agit d'un type de couche de regroupement qui réalise un regroupement maximal d'entrées de tailles non uniformes des features maps. Le choix de cette taille fixe est un hyper-paramètre réseau et est prédéfini.

L'objectif principal de cette couche est d'accélérer le processus de l'entrainement du modèle.

2.2.2 Commentaires sur le Fast RCNN :

La raison pour laquelle « Fast R-CNN » est plus rapide que R-CNN est qu'il n'est pas nécessaire d'alimenter 2000 propositions de régions pour le CNN à chaque fois. Au lieu de cela, l'opération de convolution est effectuée une seule fois par image et des feature maps sont générées à partir de celle-ci.

D'autre part, la différence entre Fast RCNN et RCNN c'est au niveau de l'input pour la proposition des régions, en effet, RCNN fait son input au niveau du pixel pour la CNN, alors que le Fast RCNN le fait au niveau des feature maps.

2.3 Faster RCNN :

2.3.1 Partie théorique du Faster RCNN :

Le vrai problème pour les 2 modèles ci-dessus, qui sont le RCNN et le Fast RCNN, c'est qu'ils utilisent l'algorithme de recherche sélective pour trouver les propositions de régions.

Comme Fast R-CNN, l'image est fournie en tant qu'entrée d'un réseau de convolution qui produit des feature maps. Ce qui change ici c'est qu'au lieu d'utiliser un algorithme de recherche sélective sur les différents feature maps pour identifier les propositions de région, un réseau séparé est utilisé pour prédire les propositions de région.

Ces propositions de régions prédites sont ensuite remodelées à l'aide d'une couche de regroupement RoI qui est ensuite utilisée pour classer l'image dans la région proposée et pour prédire les valeurs du bounding box.

Pour résumer un peu plus l'architecture, le faster RCNN se compose de 2 principaux composants :

1. RPN (propositions de région) : Donne un ensemble de rectangles basés sur une deep convolution layer.

2. Fast-RCNN Roi Pooling layer : Classer chaque proposition et affiner l'emplacement de la proposition.

Vu que nous allons implémenter Faster RCNN, nous allons entrer plus dans les détails de ce modèle. Nous montrons dans la figure qui suit, l'architecture du Faster RCNN :

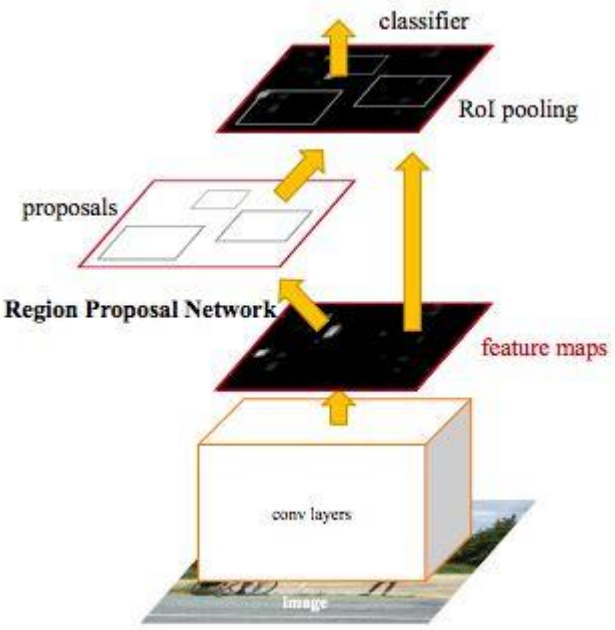

Figure 3 : Architecture Faster RCNN

De nombreuses personnes implémentent Faster R-CNN pour identifier les objets, mais cet algorithme s'intéresse particulièrement à la logique et aux calculs sous-jacents à la manière dont cet algorithme récupère les objets identifiés. En effet, on parle ici de ce que fait le RPN.

RPN génère la proposition les objets. RPN possède une architecture unique et spécialisée. Nous allons détailler plus cette architecture sur la figure qui suit :

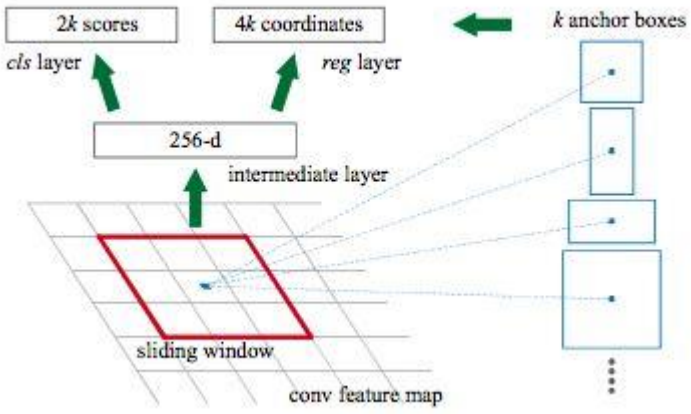

Figure 4 : Architecture RPN

RPN a un classificateur et un régresser. C'est avec les RPN que le concept d'Anchor est venu. L'Anchor est le point central de la fenêtre qui glisse.

Il est a noté également que le RPN est très robuste pour les translations, de ce fait, l'une de ces propriétés est qu'il est invariant par translation.

Passons en fait comment le RPN propose les régions :

Les différents Anchors (mentionnés sur le paragraphe ci-dessus) ont des étiquettes, ces dernières sont assignées en se basant l'IoU (Intersection Over Union).

Il ne faut pas également oublier que les RPN sont des réseaux que nous devons entrainés au préalable. Généralement, on utilise des modèles qui sont déjà entrainés tels que VGG-16, ResNet 50, DeepNet, AlexNet by ImageNet.

Pour voir les différents modèles qui peuvent être utilisé, veuillez voir ce lien :

https://github.com/tensorflow/models/tree/master/research

Faisons alors un récapitulative, sur tout le fonctionnement du Faster RCNN :

1. Au début, l'image passe à travers la couche conv, et les différents features maps sont extraites.
2. Cela étant, une fenêtre qui translate est utilisée au niveau du RPN pour les différentes localisations pour chaque feature map.
3. Pour chaque localisation, il aura (k=9) Anchors qui sont utilisés (3 échelles de 128, 256 et 512 et 3 formats de 1: 1, 1: 2, 2: 1) pour générer les régions de propositions.
4. Une couche **cls** génère 2k scores, pour savoir si un objet existe ou pas pour les k Anchors.
5. Une couche **reg** génère 4k pour les différentes coordonnées des différents bounding boxes.
6. Avec des tailles de feature maps de W * H, nous aurons WHk Anchors au total.
7. Le réseau RPN doit vérifier préalablement quel emplacement contient un objet. Après, les emplacements et les boîtes englobantes correspondants passeront au réseau de détection pour détecter la classe de l'objet et renvoyer la boîte englobante de cet objet.
8. Nous passons après au réseau de détection. Cette partie est similaire au Fast RCNN.

2.3.2 Commentaires sur le Faster RCNN :

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	**0.2 seconds**
(Speedup)	1x	25x	**250x**
mAP (VOC 2007)	66.0	**66.9**	**66.9**

Comme nous le voyons sur le tableau ci-dessus, Faster RCNN est le plus rapide des 3 modèles en prenant juste 0.2 secondes pour chaque image du test set.

Cela est expliqué par le fait que Faster RCNN n'utilise pas l'algorithme de recherche selective pour proposer les régions.

D'autre part, nous voyons que Fast RCNN et Faster RCNN ont le même mAP (mean Average Precision), ce qui montre que Faster RCNN et Fast RCNN sont les plus performants en termes de précision sur la détection des objets.

III. Implémentation du Faster RCNN :

Nous avons choisi de faire une implémentation de Faster RCNN pour illustrer ce qui a été évoqué dans l'article que nous avons choisi.

Vu que c'était difficile d'exécuter le code sur un GPU, nous avons opté pour cette approche :

1. Nous avons écrit un code python sous forme de différents fichiers .py qui implémentent le modèle Faster RCNN. Nous indiquons en détails dans le fichier **Readme.pdf** la manière avec laquelle on entraine le modèle et comment on l'applique sur l'ensemble test-set.
   Le dossier à consulter pour trouver ce travail et les différents fichiers .py est :
   Article1_Notebook_**1.**
   Il faudra tourner le code sur un gpu puissant sur le dataset PascalVOC2007.

2. Nous avons fait également un autre notebook à part, où nous avons télécharger un modèle Faster RCNN déjà entrainé (ils sont disponibles sur internet et que la plupart des personnes utilisent pour faire du transfer learning quand c'est nécessaire) pour faire de la détection et localisation d'objets sur des images que nous avons choisies.

   Les différents détails sur cette approche sont sur le notebook.

   Le notebook pour trouver les résultats et les différents classification et localisation d'objet est : **Article1_Notebook_2.ipynb** dans le dossier **Article1_Notebook_2.**

IV. Critiques sur les modèles :

4.1. Récapitulation des modèles et critiques :

Pour commencer, faisons un petit résumé sur les modèles de la famille RCNN.

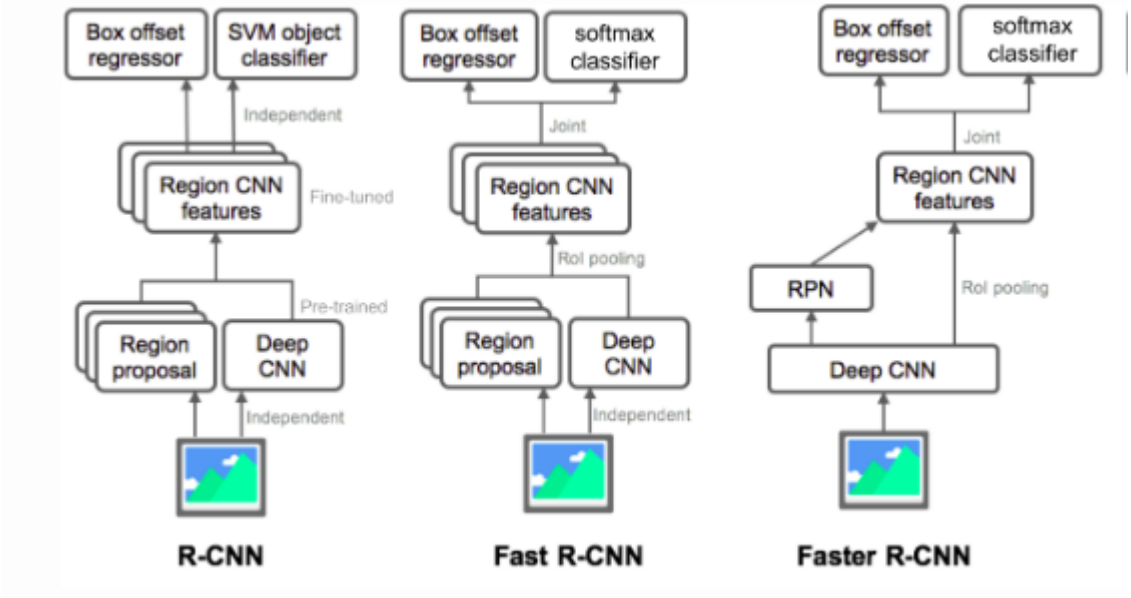

Figure 5 : Comparaison d'architecture

Il est a noté ici, que l'article n'a pas traité le **Mask-RCNN** qui est la dernière version de la famille RCNN.

Comme nous le voyons sur l'image ci-dessus, tous les algorithmes de détection d'objet précédents utilisent des régions pour localiser l'objet dans l'image (Region proposal sur les diagrammes).

Le modèle ne regarde pas l'image complète, mais plutôt les propositions de région que ce soit par l'algorithme de recherche sélective pour le RCNN et le Fast RCNN, ou bien par le RPN pour le Faster RCNN.

De ce fait, ce qui sera plus intéressant, c'est d'avoir des modèles qui traitent l'ensemble des images et non pas juste les propositions de région.

4.2. Les alternatives:

4.2.1. YOLO (You Only Look Once):

Yolo est un algorithme de détection d'objet très différent des algorithmes basés sur les propositions de régions vus ci-dessus. Pour YOLO, un seul réseau convolutionnel prédit les boîtes englobantes et les probabilités de classe pour ces boîtes.

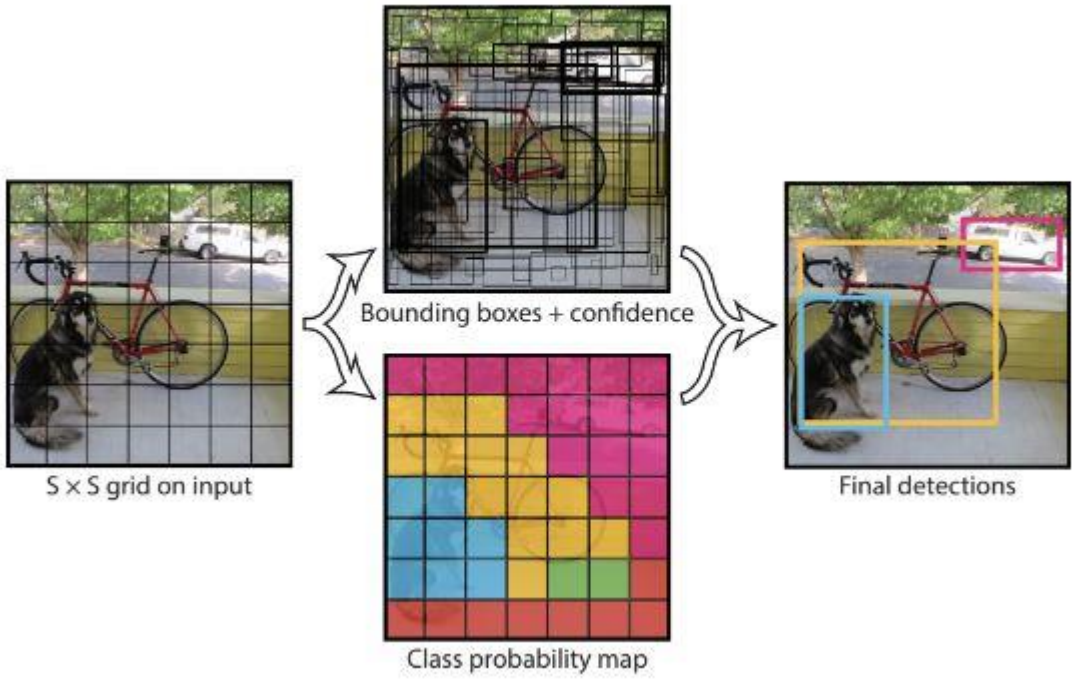

Figure 6 : YOLO

Le fonctionnement de YOLO consiste à prendre une image et à la décomposer en une grille SxS, au sein de chaque grille, nous prenons m bounding boxes. Pour chacune des boîtes englobantes, le réseau génère une probabilité de classe et des valeurs de décalage pour la boîte englobante. Les boîtes englobantes ayant la probabilité de classe supérieure à une valeur de seuil sont sélectionnées et utilisées pour localiser l'objet dans l'image.

YOLO est beaucoup plus rapide (45 images par seconde) que d'autres algorithmes de détection d'objets. La limite de l'algorithme YOLO réside dans le fait qu'il pour avoir des difficultés pour détecter les petits objets dans l'image.

4.2.2. SSD (Single Shot Detector) :

En utilisant SSD, il suffit d'une seule prise de vue pour détecter plusieurs objets dans l'image, tandis que les approches basées sur un réseau pour les propositions de régions (RPN) telles que la série R-CNN nécessitent deux prises de vues, l'une pour générer des propositions de région, l'autre pour la détection des objets de chaque proposition. Ainsi, le SSD est beaucoup plus rapide que les approches à deux plans basés sur la RPN.

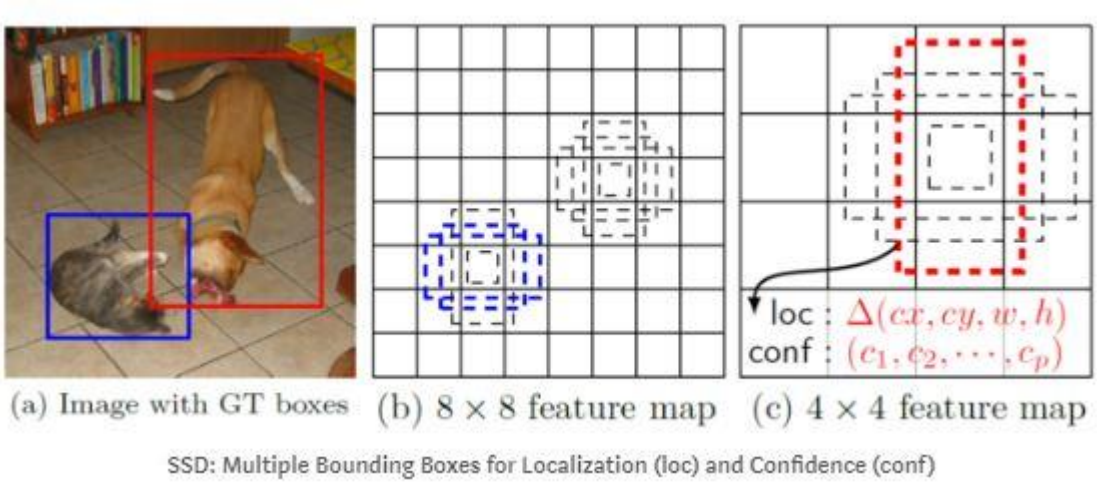

Figure 7 : SSD

Nous allons un peu expliquer le fonctionnement du SSD :

1. Après avoir parcouru certaines couches de convolutions pour l'extraction des features, nous obtenons une couche de feature de taille m × n avec p canaux, tels que 8 × 8 ou 4 × 4 ci-dessus. Et une conv 3 × 3 est appliquée sur cette couche d'entités m × n × p.
2. Pour chaque emplacement, nous avons k boîtes englobantes. Ces k boîtes englobantes ont différentes tailles et proportions. Le concept est peut-être qu'un rectangle vertical est plus adapté à l'homme et un rectangle horizontal est plus adapté à la voiture.
3. Pour chaque cadre de sélection, nous allons calculer les 'c' scores pour chaque classe et 4 valeurs par rapport à la forme du cadre de sélection par défaut.
4. On aurons alors, (c+4)*k*m*n résultats.

4.3.3. Résultats :

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19

Figure 8 : Les résultats

Après avoir entrainé les différents modèles sur le fameux dataset Pascal VOC 2007 + 2012. Il y a eu une comparaison entre les différents modèles :

Ce que nous pouvons tirer du tableau ci-dessus, c'est qu'en termes de mAP (mean Average Precision), SSD a eu le meilleur score avec 76.8. Il est a noté également que Faster RCC avec ResNet a eu un mAP un peu similaire de l'ordre de 76.4.

D'autre part, il faut tenir compte également de FPS (Frame Per Second), nous remarquons que Faster RCNN est trop lent par rapport aux autres modèles (nous l'avons expliqué dans le rapport).

En termes de rapport mAP par FPS, SSD300 est le meilleur avec un mAP de 74.3 et un FPS de 46 images par seconde.